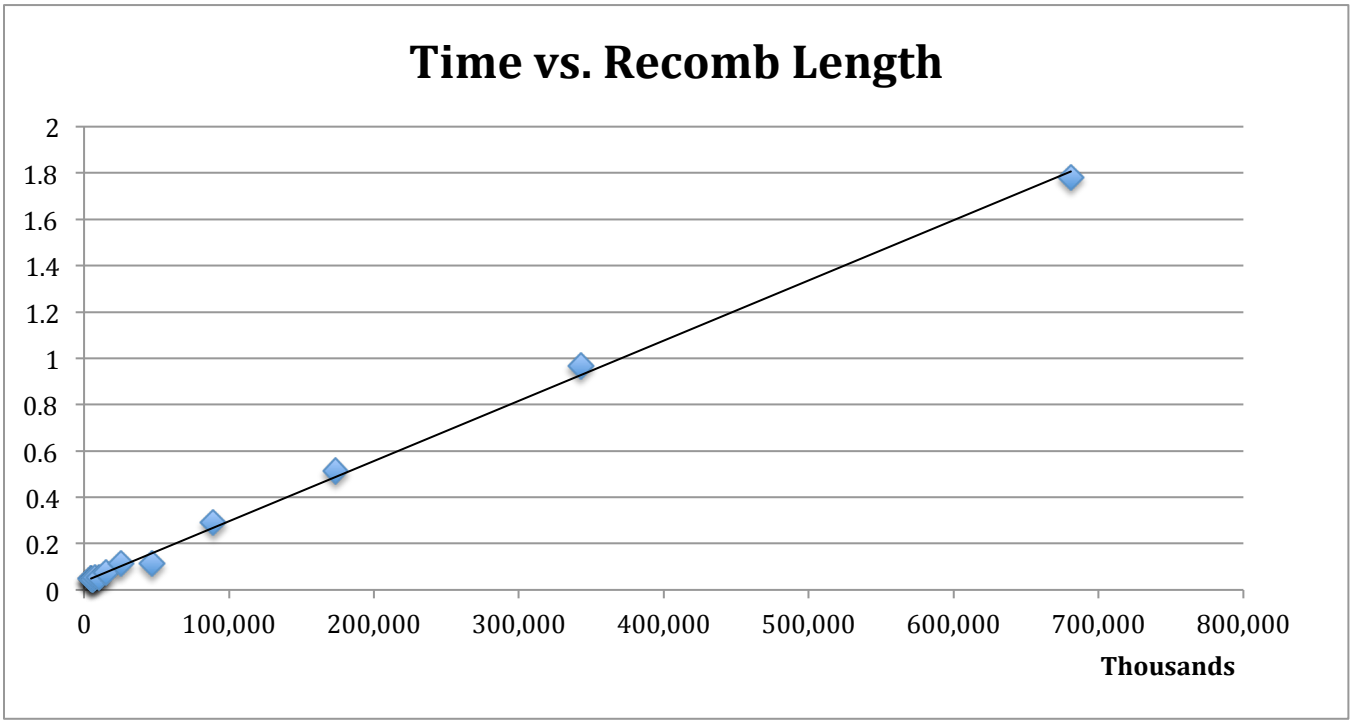
Parker Pearson
Analysis
Part 1a:



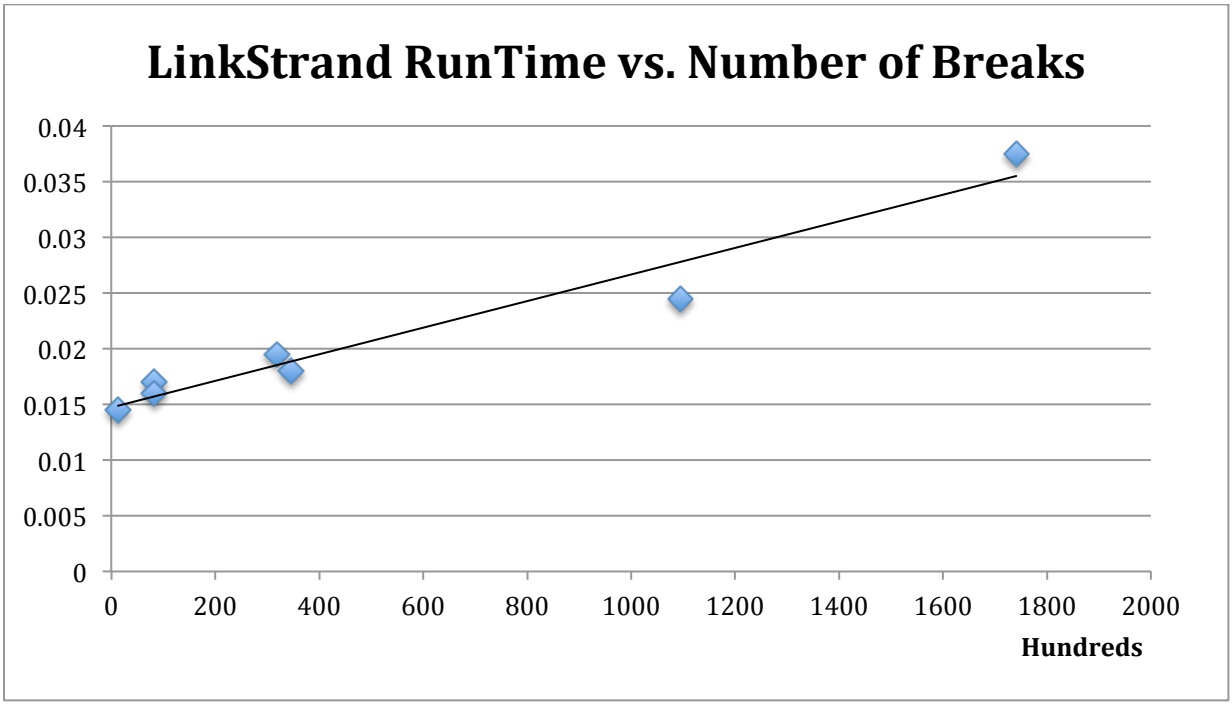
Using DNABenchmark.java the above data was acquired. The program calculates the time for the splice and cut in SimpleStrand given various sized splicee's. From the above figure it becomes evident that the method runs in O(n) time. This is because there is a linear relationship between the amount of input into the method and the time it takes to complete. In order to get a better representation of this growth I increased the heap space allowing for the method to handle higher magnitudes of splicee's. The method runs in O(n) time where n is the length of the string because the method must search the entire string once to find all the enzymes which takes O(n) time. Replacing each enzyme with a splicee takes a constant time or O(1), so the overall time complexity of the method is simply O(n).

Data from graph:

4,800,471	0.05
4,965,591	0.05
5,295,831	0.045
5,956,311	0.047
7,277,271	0.051
9,919,191	0.052
15,203,031	0.075
25,770,711	0.114

46,906,071	0.112
89,176,791	0.29
173,718,231	0.513
342,801,111	0.966
680,966,871	1.781

1b:
In order to determine the power-of-two string that can fit in a heap size I ran DNABenchmark.java with the specified heap size and noted the largest string able to run and the time it took. For 512 MB the heap size allowed for a string of length 89,176,791 and took .250 seconds to complete. For 1024 MB the heap size allowed for a string for length 173,718,231 and took .518 seconds to complete. Ecoli.txt was used as data for parts a and b.



The graph above shows that the cutAndSplice method runs in O(B) where B is the number of breaks. I acquired the data by running DNABenchmark with various enzymes which resulted in a change in number of appends. I then converted the amount of appends to the amount of breaks and plotted the recorded runtime vs. this number of breaks. The linear relationship makes sense because the inner methods becides the iteration of the nodes are all O(1). This includes various arithmetic and the substring command. Nevertheless, the method has to call upon

the append method a proportional amount of times to the amount of breaks it has. This results in the method being in O(B) time where B is the number of breaks.

Data from graph:

1290	0.0145
8320	0.017
34638	0.018
109474	0.0245
174062	0.0375
31764	0.0195
8312	0.016

I have upheld the Duke Community Standard in completing this assignment